Technical specification for the feature: Forms editor

This is the technical specification for the Forms editor feature. The feature was created as part of an EU-funded project called We4Authors Cluster. All features, their technical specification and their video documentation are published on www.accessibilitycluster.com.

If you have any questions or comments, please do not hesitate to reach out to research@funka.com.

The technical specification contains several perspectives:

- the web interface of the feature, based on the experience of the end user or, when it comes to the testing-features, the web author;
- the code of the feature ensuring accessibility by default;
- a description highlighting the key elements of the code;
- reference to a video documentation;
- recommendations for implementation.

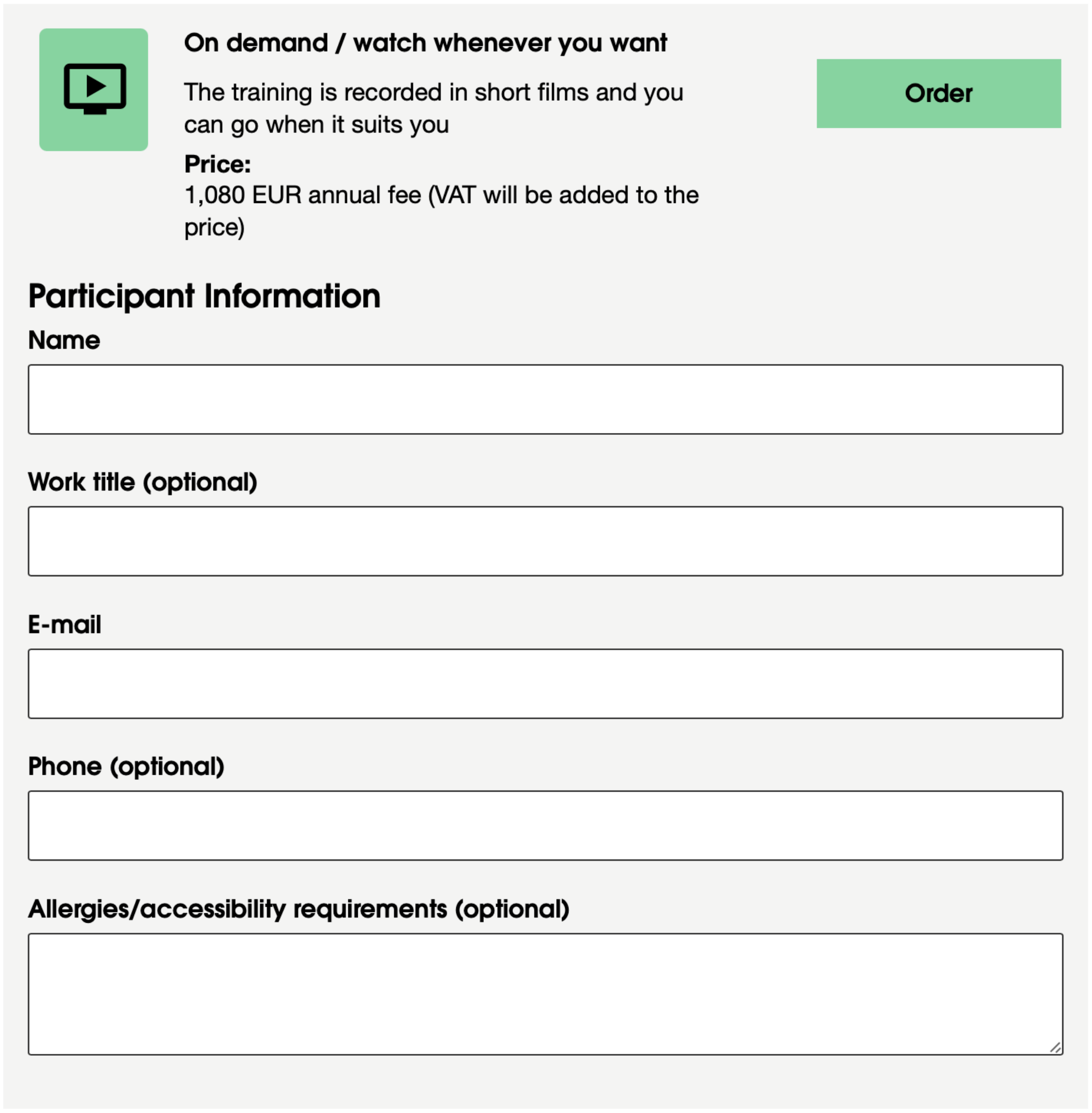
Specifications of the feature: Forms editor

Accessibility by default or built-in support for accessible content creation.

A forms editor is used to create different kinds of forms that may be used for registration, contact or any other interaction with the user of the website. Forms can be very simple (fill in your email address to subscribe to our newsletter) or extremely complex, with back-end logic to save content or present material from other systems, validation or authentication as well as many pages and steps. Because the form requires interaction with the user, accessibility is key. At the same time, forms require both output and input to work in a correct way, and there is no surprise that forms are the objects where most accessibility issues are normally found.

For many users, forms can be a challenge. This is true for users with cognitive impairments, motor impairments and users with any kind of assistive technology, whether it is a screen reader, an enlargement tool, or an input device.

Web interface: the end user view



On demand / watch whenever you want

The training is recorded in short films and you can go when it suits you

Price:
1,080 EUR annual fee (VAT will be added to the price)

Participant Information

Name

Work title (optional)

E-mail

Phone (optional)

Allergies/accessibility requirements (optional)

Figure 1: End user view

Code of the end user view

```
<form action="/en/education"  method="post">

  <fieldset>
    <legend>Participant Information</legend>
    <label for="f1">Name
      <span class="errormsg"></span>
    </label>
    <input autocomplete="name" id="f1"
```

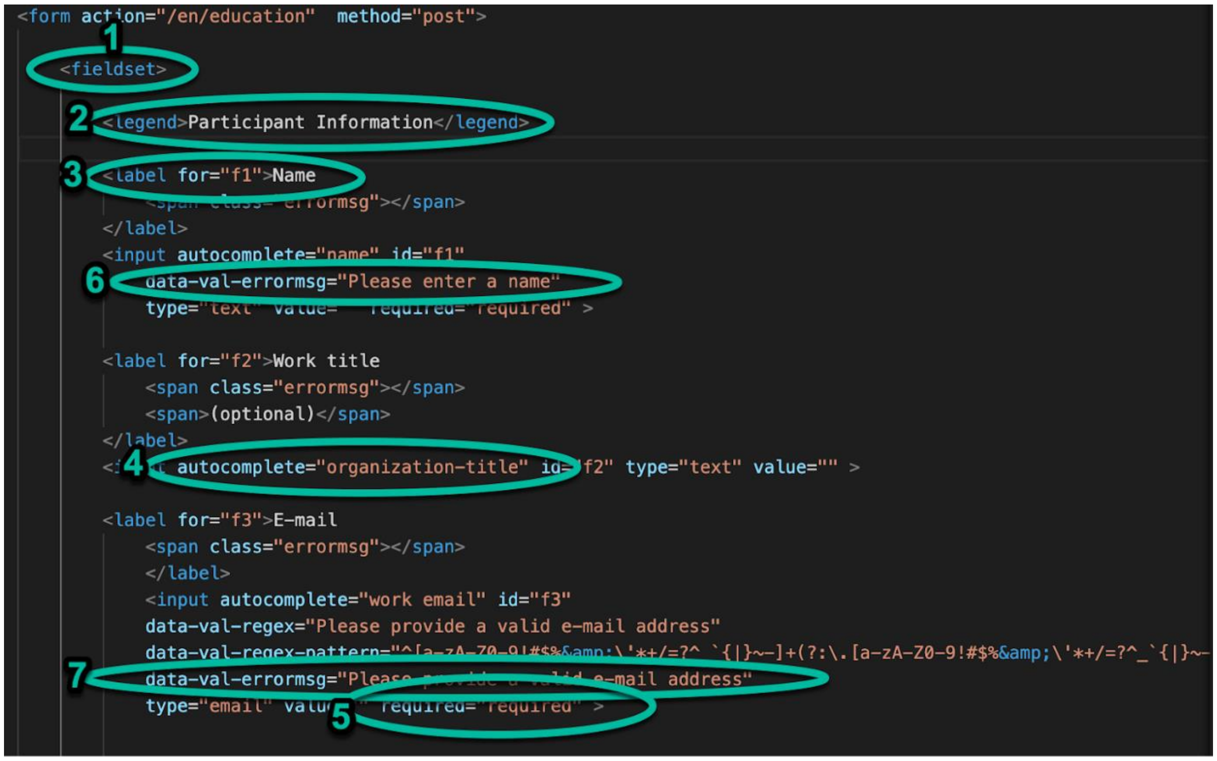
```
        data-val-errormsg="Please enter a name"
        type="text" value="" required="required" >

    <label for="f2">Work title
        <span class="errormsg"></span>
        <span>(optional)</span>
    </label>
    <input autocomplete="organization-title" id="f2" type="text" value="" >

    <label for="f3">E-mail
        <span class="errormsg"></span>
        </label>
        <input autocomplete="work email" id="f3"
        data-val-regex="Please provide a valid e-mail address"
        data-val-regex-pattern="^[a-zA-Z0-9!#$%&amp;\'*+/=?^_`{|}~-]+(?:\.[a-zA-Z0-
9!#$%&amp;\'*+/=?^_`{|}~-]+)*@(?:[a-zA-Z0-9](?:[a-zA-Z0-9-]*[a-zA-Z0-9])?\.)+[a-zA-Z0-9](?:[a-zA-Z0-9-
]*[a-zA-Z0-9])?"
        data-val-errormsg="Please provide a valid e-mail address"
        type="email" value="" required="required" >
```

Figure 2: Code example

Explanation of the code example



```
<form action="/en/education"  method="post">
  <fieldset>
    <legend>Participant Information</legend>

    <label for="f1">Name
      <span class="errormsg"></span>
    </label>
    <input autocomplete="name" id="f1"
        data-val-errormsg="Please enter a name"
        type="text" value="" required="required" >

    <label for="f2">Work title
        <span class="errormsg"></span>
        <span>(optional)</span>
    </label>
    <input autocomplete="organization-title" id="f2" type="text" value="" >

    <label for="f3">E-mail
        <span class="errormsg"></span>
        </label>
        <input autocomplete="work email" id="f3"
        data-val-regex="Please provide a valid e-mail address"
        data-val-regex-pattern="^[a-zA-Z0-9!#$%&amp;\'*+/=?^_`{|}~-]+(?:\.[a-zA-Z0-9!#$%&amp;\'*+/=?^_`{|}~-
        data-val-errormsg="Please provide a valid e-mail address"
        type="email" value="" required="required" >
```

Figure 3: Code example with highlighted elements

1. The <fieldset> tag is used to group related elements in a form.

2. The <legend> tag is used to define a caption for the <fieldset> element.

3. The <label> tag defines a label for several elements, for example:

   <input type="checkbox">
   <input type="email">
   <input type="text">
   <select>
   <textarea>

   Proper use of labels with the elements above will benefit:

   - Screen reader users (will read out aloud the label, when the user is focused on the element)
   - Users who have difficulty clicking on very small regions (such as checkboxes) - because when a user clicks the text within the <label> element, it toggles the input (this increases the hit area).

   Note: The for attribute of <label> must be equal to the id attribute of the related element to bind them together. A label can also be bound to an element by placing the element inside the <label> element.

4. The HTML autocomplete attribute is available on <input> elements that take a text or numeric value as input, <textarea> elements, <select> elements, and <form> elements. Autocomplete lets web developers specify what, if any, assistance the user agent has to provide automated assistance in filling out form field values, as well as guidance to the browser as to the type of information expected in the field.

5. The attribute required specifies that an input field must be filled out before submitting the form.

6 & 7. Provide feedback to users about the results of their form submission, whether successful or not. This includes in-line feedback at or near the form controls and overall feedback that is typically provided after form submission.

Notifications have to be concise and clear. In particular, error messages should be easy to understand and should provide simple instructions on how they can be resolved. Success messages are also important to confirm task completion.

Form editors in authoring tools should be built to assume accessibility best practices for both the author and the end-user. Default options should be set up to be semantically correct.

Video documentation

This technical specification has been developed in the We4Author Cluster project to reflect recommendations for accessibility features that can be implemented in any authoring tool. The specifications are complemented by a video documentation, covering a live description of:

- end user problems,

- web author challenges,
- feature solutions, and
- resulting accessibility for end users when the features are correctly used.

Recommendations for implementation

To make sure the implementation of the features is not causing accessibility problems for web authors with disabilities:

- avoid drag-and-drop for choosing template, and/or always have a keyboard alternative;
- always provide one pointer alternative without specific gestures;
- always support keyboard navigation;
- consider keyboard shortcuts;
- do not rely on sensory characteristics as the sole indicator for understanding and operating content;
- do not indicate important information using colour alone;
- make sure there is enough contrast between text objects and its background colour.